Claims

[c1] What is claimed is:

1. A method for architecture for a multitasking operating system, the method comprising the steps of:

Having a processing Kernel;

Controlling the execution of each task with a task control block with every task comprises one block and with a plurlarity of task control blocks arranged in a dynamic chain;

Controlling events associated with a task using a Task Information Block with every task comprising one block and with a plurlarity of Task Information Blocks arranged in a dynamic chain; .

[c2]

Controlling events associated with a task using an Event Control Block with every event comprising one block and with a plurlarity of Event Control Blocks arranged in a dynamic chain;

Controlling a task's communication pipes with a Pipe Control Block with every pipe comprising one block and with a plurlarity of Pipe Control Blocks arranged in a dynamic chain;

Managing memory assigned to a task with a Data Memory Control Block with a plurlarity of Data Memory Control Blocks arranged in a chain;

Having Task Data Memory blocks which are data memory blocks associated with each task in which a task's Task Data Memory block stores its context on one Task Data Memory block before yielding computer means handling to the Kernel;

Using Port Information Blocks wihch are special memory blocks used to handle a microprocessor mean"s input and output ports;

Having the Kernel use Kernel Control Registers which are used to store required operation data;

Having a plurlarity of Task Allocation Tables which is a table of pointers that accurately identify the beginning of the memory blocks associated with a task;

Starting a task's program code section with a Task Header which provides basic information about the task;

Having a Port State Update control block subprocess which carries out reading

and verification of a microprocessor"s input ports to set up port events, which can be signaled when the state of an input port changes;

Having a Task State Update control block subprocess which carries out reading and verification of events to update the state of tasks on the WAIT state and switch them to the READY state if their expected event is signaled;

Having a Priority Task Ordering subprocess which selects a task in the READY state that is to be switched to the ACTIVE state; and

Having a Context and Control Restore subprocess which executes the restoration of all context variables associated with the active task, and yields control of the computer means to a Parser, which continues the execution of the active task's instructions.

[c3]

2. A method for architecture for a multitasking operating system, the method comprising the steps of:

a) Having a processing Kernel;

b)Controlling the execution of each task with a task control block with every task comprises one block and with a plurlarity of task control blocks arranged in a dynamic chain and containg the following fields;

i) Task Identifier which contains the identifier of the task and initialized when the task is installed into the Kernel"s line of execution;

ii) Status/Priority field which contains the task's state and priority information and is fixed when the task is installed on the Kernel"s line of execution;

iii) a Task Next Instruction Pointer which stores a pointer to Program Memory where the next instruction associated with the task and updated every time a task yield control to the Kernel;

iv) a Task Start Instruction Pointer that contains the pointer to Program Memory where the first instruction associated with the task and is installed in the Kernel"s line of execution with the value indicated by the Task Header;

v) a Ready_Wait_Time field which is the task waiting time in Ready State which value is updated by the Kernel;

vi) an Event Identifier that contains the event identifier for which a task is waiting to switch from the WAIT to the READY state and which is updated when the task waits for an event;

vii) an Event Control field that contains the event control value that shows that an event is active and is updated when a task starts its wait for an event;
viii) a Next Task Control Block Pointer that contains the pointer to Data Memory of the next Task Control Block in the chain of Task Control Blocks, if the value is NULL, it means that there are not other Task Control Blocks on the chain.

[c4] c) Controlling events associated with a task using a Task Information Block with every task comprising one block and with a plurlarity of Task Information Blocks arranged in a dynamic chain and with containing the following fields;
i) a Task Identifier that contains the identifier to the corresponding task and is set when the task is installed in the Kernel"s line of execution;
ii) a Pipe Identifier that contains a pointer to the Pipe that a task uses for communication purposes, if it has a NULL value it means the task has not been assigned a Pipe and is initialized when a task requests a Pipe;
iii) a Mutual Exclusion Identifier contains the identifier of the mutually exclusive event that a task owns, if the value is NULL then no mutually exclusive event has been assigned to the task and it is initialized when a task requests a mutually exclusive event; and
iv) a Next Task Information Block Pointer that contain a data memory pointer to the next Task Information Block Task Information Block chain, if the value is NULL, it means no next Task Information Block exists in the chain and it is updated by the Kernel when the current next Task Information Block is the last on the next Task Information Block chain and a new next Task Information Block is created.

[c5] d) Controlling events associated with a task using a Event Control Block with every event comprising one block and with a plurlarity of Event Control Blocks arranged in a dynamic chain and with the following fields;
i) an Event Identifier that contains the identifier of the event being expected by a task so that it can switch from the WAIT to the READY task state and is initialized when an event is created;
ii) an event"s Control & Status field contains information about the event, it is updated by the task that controls the event and is watched by the Kernel to decide whether an event has been signaled by comparing it with the event"s

Event_Control field within the Task Control Block;

iii) a Next Event Control Block Pointer that contain the data memory pointer to the next Event Control Block in the Event Control Block chain and it is updated by the Kernel when the current Event Control Block is the last on the Event Control Block chain and a new Event Control Block is created.

[c6] e) Controlling a task's communication pipes with a Pipe Control Block with every pipe comprising one block and with a plurlarity of Pipe Control Blocks arranged in a dynamic chain;

i) a Pipe Identifier which contains the identifier of the Pipe created on memory;

ii) a Pipe Data Memory Base Address contain a data memory pointer to the first byte on the Pipe and is initialized when the Pipe is created;

iii) a Pipe Size field contains the number of data memory bytes that a Pipe occupies and is initialized when the Pipe is created; and

iv) a Next Pipe Control Block Pointer field contains a data memory pointer to the next Pipe Control Block in the chain of Pipe Control Blocks, if this field is NULL, it means there is no next Pipe Control Block on the chain and the value is updated by the Kernel when the current Pipe Control Block is the last on the Pipe Control Block chain a new Pipe Control Block is created.

[c7] f) Managing memory assigned to a task with a Data Memory Control Block with a plurlarity of Data Memory Control Blocks arranged in a chain and containing the following fieds;

i) a Task Identifier field which contains the identifier of the task, it is initialized when the block is created when the task is installed on the Kernel"s line of execution;

ii) a Task Data Memory Base Address is a data memory pointer to the first byte in the memory block assigned to a task and are initialized when Task Memory Data is assigned; and

iii) Task Data Memory Size which contains the number of data memory bytes of the assigned memory block and is initialized when the block is created;

iv) a Next Task Data Memory Control Block Pointer is a data memory pointer to the next Data Memory Control Block in the Data Memory Control Block chain, if it has a NULL value, it means no next Data Memory Control Block exists and is

updated by the kernel when the current Data Memory Control Block is the last Data Memory Control Block on the Data Memory Control Block chain and a new Data Memory Control Block is created.

[c8] g) Having a Task Data Memory blocks which are data memory blocks associated with each task in which a task's Task Data Memory block stores its context on one Task Data Memory block before yielding computer means handling to the Kernel and has the following fields;

i) a Task Identifier contains the identifier of the task and is initialized when the block is created when the task is installed on the Kernel"s line of execution;

ii) a plurlarity of task context registers; and

iii) a plurlarity of task data registers.

[c9] j) Having Port Information Blocks which are memory blocks used to handle a microprocessor mean"s input and output ports and containing the following fields;

i) an Assignation_Port_Mask field that contains information of microprocessor mean"s inputs/outputs ports assigned to tasks;

ii) a Input_Selection_Mask contains information of the direction of communications in inputs/outputs ports; and

iii) a Change_State field that contains information related to a change in the value of the microprocessor"s inputs/outputs bits.

[c10] k) Having the Kernel use Kernel Control Registers which are used to store required operation data and containing the following fields;

i) a Free Data Memory Remainder field that contains the number of free bytes in data memory;

ii) a Max_Priority register that stores the maximum priority assigned to a task used in the process of searching for the READY task with highest priority;

iii) a Max_Wait_Time register that stores the maximum wait time associated with a task and used in the process of searching for the READY task with the highest priority;

iv) a Task_ID_Winner Identifier which is the READY task of highest priority;

v) an Instruction Pointer to program memory that contains the current

instruction that is to be decoded by the Parser;

vi) a Data Memory Pointer;

vii) a Task Control Block Pointer which is a data memory pointer to the location of the current Task Control Block;

viii) an Event Control Block Pointer which is a data memory pointer to the location of the current Event Control Block;

ix) a Data Memory Control Block Pointer which is a data memory pointer to the location of the current Data Memory Control Block;

x) a Task Information Block Pointer which is a data memory pointer to the current Task Information Block;

xi) a Pipe Control Block Pointer which is a data memory pointer to the location of the current Pipe Control Block;

xii) a Last Task Control Block Pointer which is a data memory pointer to the location of the last Task Control Block in the Task Control Block chain;

xiii) a Last Event Control Block Pointer which is a data memory pointer to the location of the last Event Control Block in the Event Control Block chain;.

[c11] xiv) a Last Task Information Block Pointer which is a data memory pointer to the location of the last Task Information Block on the Task Information Block chain;

xv) a Last Pipe Control Block Pointer which is a data memory pointer to the location of the last Pipe Control Block on the Pipe Control Block chain; and

xvi) a Last Data Memory Control Block Pointer which is a data memory pointer to the location of the last Data Memory Control Block in the Data Memory Control Block chain.

[c12] l) Having a plurlarity of Task Allocation Table which are tables of pointers that accurately identify the beginning of the memory blocks associated with a task and containing the following fields;

i) a Task_Condition field which indicates the initial operation condition associated with the task that a Task Allocatoin Table entry points with the possible conditions of SLEEP, to indicate that the task is not to be inserted in the Kernel"s line of execution at system start time and WAKE, to indicate that the task must be installed on the Kernel"s line of execution upon system startup; and.

[c13] ii) a Task Start Instruction Pointer which is a program memory pointer to the location of a task's Task Header.

[c14] k) Starting a task's program code section with a Task Header which provides basic information about the task and contains the following fields;
i) a Task Identifier contains the identifier of the task; .

[c15] ii) a Status/Priority field that contains the task's state information and assigned priority;
iii) a Task Data Memory Size field that contains the number of bytes that the assigned data memory block occupies;. and
iv) the Task Program Code which is the task's program code that is to be executed by the Kernel.

[c16] m) Having a Port State Update control block subprocess which carries out reading and verification of a microprocessor"s input ports to set up port events, which can be signaled when the state of an input port changes;
n) Having a Task State Update control block subprocess which carries out reading and verification of events to update the state of tasks on the WAIT state and switch them to the READY state if their expected event is signaled including the following steps:
i) having the pointer to the beginning of the Task Control Block chain retrieved with all the Task Control Blocks in the Task Control Block chain and examining the state associated with every task;
ii) Checking to see if a task is in the SUSPEND state, if it is then its state is not modified in this subprocess and the task control block pointer points to the next link in the Task Control Block chain if the next link is NULL then it is the end of the chain;
iii) Checking to see if a task is in READY state, if it is this means that the current task is not expecting any events;
iv) Checking to see if a task is in WAIT state, if it is then the associated event must be examined, the pointer to the beginning of the Event Control Block chain is retrieved and examine until the expected event is found, if the current Event Control Block corresponds to the event, its state is examined, if it is

activated then the state of the task in WAIT state is changed to a READY state, if the end of the Event Control Block chain is reached without finding the event, a system error is flagged.

[c17] o) Having a Priority Task Ordering subprocess which selects the task in the READY state that is to be switched to the ACTIVE state using the follow criteria the task priority and the amount of time in which a task has been in the READY state using the following steps:
i) retreiving the pointer from the beginning of the Task Control Block;
ii) examining all the tasks in the Task Control Block that are in the READY state;.

[c18] iii) intializing the max_priority, max_wait_time and task_id_winnner fields;
iv) checking sequentially all task states, if a task is not READY, the next task in the chain is checked, if a task is READY its priority is checked against Max_Priority, if the task's priority is higher, store task's data in the Max_Priority, Max_Wait_Time and Task_ID_Winner; if the task's priority is equal to the Max_Priority field, then compare the task's time on the WAIT state to the Max_Wait_Time field, if the current task's time on the WAIT state is higher, store task's data in the Max_Priority, Max_Wait_Time and Task_ID_Winner; and
v) repeating the prevoius step until the last link in the Task Control Block chain is examined. and
Having a Context and Control Restore subprocess which executes the restoration of all context variables associated with the active task and yields computer means control to a Parser, which continues the execution of the active task's instructions.

[c19]

3. A computer program wherein the base component has interfaces and the program code for:
Having a processing Kernel;
Controlling the execution of each task with a task control block with every task comprises one block and with a plurlarity of task control blocks arranged in a dynamic chain;
Controlling events associated with a task using a Task Information Block with every task comprising one block and with a plurlarity of Task Information Blocks

arranged in a dynamic chain; .

[c20]

Controlling events associated with a task using an Event Control Block with every event comprising one block and with a plurlarity of Event Control Blocks arranged in a dynamic chain;

Controlling a task's communication pipes with a Pipe Control Block with every pipe comprising one block and with a plurlarity of Pipe Control Blocks arranged in a dynamic chain;

Managing memory assigned to a task with a Data Memory Control Block with a plurlarity of Data Memory Control Blocks arranged in a chain;

Having Task Data Memory blocks which are data memory blocks associated with each task in which a task's Task Data Memory block stores its context on one Task Data Memory block before yielding computer means handling to the Kernel;

Using Port Information Blocks which are special memory blocks used to handle a microprocessor mean"s input and output ports;

Having the Kernel use Kernel Control Registers which are used to store required operation data;

Having a plurlarity of Task Allocation Table which is a table of pointers that accurately identify the beginning of the memory blocks associated with a task;

Starting a task's program code section with a Task Header which provides basic information about the task;

Having a Port State Update control block subprocess which carries out reading and verification of a microprocessor"s input ports to set up port events, which can be signaled when the state of an input port changes;

Having a Task State Update control block subprocess which carries out reading and verification of events to update the state of tasks on the WAIT state and switch them to the READY state if their expected event is signaled;

Having a Priority Task Ordering subprocess which selects a task in the READY state that is to be switched to the ACTIVE state; and

Having a Context and Control Restore subprocess which executes the restoration of all context variables associated with the active task, and yields control of the computer means to a Parser, which continues the execution of the

active task's instructions.

[c21] 4. A method for architecture for a multitasking operating system, the method comprising the steps of:

a) Having a processing Kernel;

b)Controlling the execution of each task with a task control block with every task comprises one block and with a plurlarity of task control blocks arranged in a dynamic chain and containg the following fields;

i) Task Identifier which contains the identifier of the task and initialized when the task is installed into the Kernel"s line of execution;

ii) Status/Priority field which contains the task's state and priority information and is fixed when the task is installed on the Kernel"s line of execution;

iii) a Task Next Instruction Pointer which stores a pointer to Program Memory where the next instruction associated with the task and updated every time a task yield control to the Kernel;

iv) a Task Start Instruction Pointer that contains the pointer to Program Memory where the first instruction associated with the task and is installed in the Kernel"s line of execution with the value indicated by the Task Header;

v) a Ready_Wait_Time field which is the task waiting time in Ready State which value is updated by the Kernel;

vi) an Event Identifier that contains the event identifier for which a task is waiting to switch from the WAIT to the READY state and which is updated when the task waits for an event;

vii) an Event Control field that contains the event control value that shows that an event is active and is updated when a task starts its wait for an event;

viii) a Next Task Control Block Pointer that contains the pointer to Data Memory of the next Task Control Block in the chain of Task Control Blocks, if the value is NULL, it means that there are not other Task Control Blocks on the chain.

[c22] c) Controlling events associated with a task using a Task Information Block with every task comprising one block and with a plurlarity of Task Information Blocks arranged in a dynamic chain and with containing the following fields;

i) a Task Identifier that contains the identifier to the corresponding task and is set when the task is installed in the Kernel"s line of execution;

ii) a Pipe Identifier that contains a pointer to the Pipe that a task uses for communication purposes, if it has a NULL value it means the task has not been assigned a Pipe and is initialized when a task requests a Pipe;
iii) a Mutual Exclusion Identifier contains the identifier of the mutually exclusive event that a task owns, if the value is NULL then no mutually exclusive event has been assigned to the task and it is initialized when a task requests a mutually exclusive event; and
iv) a Next Task Information Block Pointer that contain a data memory pointer to the next Task Information Block Task Information Block chain, if the value is NULL, it means no next Task Information Block exists in the chain and it is updated by the Kernel when the current next Task Information Block is the last on the next Task Information Block chain and a new next Task Information Block is created.

[c23] d) Controlling events associated with a task using a Event Control Block with every event comprising one block and with a plurlarity of Event Control Blocks arranged in a dynamic chain and with the following fields;
i) an Event Identifier that contains the identifier of the event being expected by a task so that it can switch from the WAIT to the READY task state and is initialized when an event is created;
ii) an event"s Control & Status field contains information about the event, it is updated by the task that controls the event and is watched by the Kernel to decide whether an event has been signaled by comparing it with the event"s Event_Control field within the Task Control Block;
iii) a Next Event Control Block Pointer that contain the data memory pointer to the next Event Control Block in the Event Control Block chain and it is updated by the Kernel when the current Event Control Block is the last on the Event Control Block chain and a new Event Control Block is created.

[c24] e) Controlling a task's communication pipes with a Pipe Control Block with every pipe comprising one block and with a plurlarity of Pipe Control Blocks arranged in a dynamic chain;
i) a Pipe Identifier which contains the identifier of the Pipe created on memory;
ii) a Pipe Data Memory Base Address contain a data memory pointer to the first

byte on the Pipe and is initialized when the Pipe is created;

iii) a Pipe Size field contains the number of data memory bytes that a Pipe occupies and is initialized when the Pipe is created; and

iv) a Next Pipe Control Block Pointer field contains a data memory pointer to the next Pipe Control Block in the chain of Pipe Control Blocks, if this field is NULL, it means there is no next Pipe Control Block on the chain and the value is updated by the Kernel when the current Pipe Control Block is the last on the Pipe Control Block chain a new Pipe Control Block is created.

[c25] f) Managing memory assigned to a task with a Data Memory Control Block with a plurlarity of Data Memory Control Blocks arranged in a chain and containing the following fieds;

i) a Task Identifier field which contains the identifier of the task, it is initialized when the block is created when the task is installed on the Kernel"s line of execution;

ii) a Task Data Memory Base Address is a data memory pointer to the first byte in the memory block assigned to a task and are initialized when Task Memory Data is assigned; and

iii) Task Data Memory Size which contains the number of data memory bytes of the assigned memory block and is initialized when the block is created;

iv) a Next Task Data Memory Control Block Pointer is a data memory pointer to the next Data Memory Control Block in the Data Memory Control Block chain, if it has a NULL value, it means no next Data Memory Control Block exists and is updated by the kernel when the current Data Memory Control Block is the last Data Memory Control Block on the Data Memory Control Block chain and a new Data Memory Control Block is created.

[c26] g) Having a Task Data Memory blocks which are data memory blocks associated with each task in which a task's Task Data Memory block stores its context on one Task Data Memory block before yielding computer means handling to the Kernel and has the following fields;

i) a Task Identifier contains the identifier of the task and is initialized when the block is created when the task is installed on the Kernel"s line of execution;

ii) a plurlarity of task context registers; and

iii) a plurlarity of task data registers.

[c27] j) Having Port Information Blocks which are memory blocks used to handle a microprocessor mean"s input and output ports and containing the following fields;

i) an Assignation_Port_Mask field that contains information of microprocessor mean"s inputs/outputs ports assigned to tasks;

ii) a Input_Selection_Mask contains information of the direction of communications in inputs/outputs ports; and

iii) a Change_State field that contains information related to a change in the value of the microprocessor"s inputs/outputs bits.

[c28] k) Having the Kernel use Kernel Control Registers which are used to store required operation data and containing the following fields;

i) a Free Data Memory Remainder field that contains the number of free bytes in data memory;

ii) a Max_Priority register that stores the maximum priority assigned to a task used in the process of searching for the READY task with highest priority;

iii) a Max_Wait_Time register that stores the maximum wait time associated with a task and used in the process of searching for the READY task with the highest priority;

iv) a Task_ID_Winner Identifier which is the READY task of highest priority;

v) an Instruction Pointer to program memory that contains the current instruction that is to be decoded by the Parser;

vi) a Data Memory Pointer;

vii) a Task Control Block Pointer which is a data memory pointer to the location of the current Task Control Block;

viii) an Event Control Block Pointer which is a data memory pointer to the location of the current Event Control Block;

ix) a Data Memory Control Block Pointer which is a data memory pointer to the location of the current Data Memory Control Block;

x) a Task Information Block Pointer which is a data memory pointer to the current Task Information Block;

xi) a Pipe Control Block Pointer which is a data memory pointer to the location

of the current Pipe Control Block;

xii) a Last Task Control Block Pointer which is a data memory pointer to the location of the last Task Control Block in the Task Control Block chain;

xiii) a Last Event Control Block Pointer which is a data memory pointer to the location of the last Event Control Block in the Event Control Block chain;.

[c29] xiv) a Last Task Information Block Pointer which is a data memory pointer to the location of the last Task Information Block on the Task Information Block chain;

xv) a Last Pipe Control Block Pointer which is a data memory pointer to the location of the last Pipe Control Block on the Pipe Control Block chain; and

xvi) a Last Data Memory Control Block Pointer which is a data memory pointer to the location of the last Data Memory Control Block in the Data Memory Control Block chain.

[c30] l) Having a plurlarity of Task Allocation Table which are tables of pointers that accurately identify the beginning of the memory blocks associated with a task and containing the following fields;

i) a Task_Condition field which indicates the initial operation condition associated with the task that a Task Allocatoin Table entry points with the possible conditions of SLEEP, to indicate that the task is not to be inserted in the Kernel"s line of execution at system start time and WAKE, to indicate that the task must be installed on the Kernel"s line of execution upon system startup; and.

[c31] ii) a Task Start Instruction Pointer which is a program memory pointer to the location of a task's Task Header.

[c32] k) Starting a task's program code section with a Task Header which provides basic information about the task and contains the following fields;

i) a Task Identifier contains the identifier of the task; .

[c33]

ii) a Status/Priority field that contains the task's state information and assigned priority;

iii) a Task Data Memory Size field that contains the number of bytes that the assigned data memory block occupies;. and

iv) the Task Program Code which is the task's program code that is to be executed by the Kernel.

[c34] m) Having a Port State Update control block subprocess which carries out reading and verification of a microprocessor"s input ports to set up port events, which can be signaled when the state of an input port changes;

n) Having a Task State Update control block subprocess which carries out reading and verification of events to update the state of tasks on the WAIT state and switch them to the READY state if their expected event is signaled including the following steps:

i) having the pointer to the beginning of the Task Control Block chain retrieved with all the Task Control Blocks in the Task Control Block chain and examining the state associated with every task;

ii) Checking to see if a task is in the SUSPEND state, if it is then its state is not modified in this subprocess and the task control block pointer points to the next link in the Task Control Block chain if the next link is NULL then it is the end of the chain;

iii) Checking to see if a task is in READY state, if it is this means that the current task is not expecting any events;

iv) Checking to see if a task is in WAIT state, if it is then the associated event must be examined, the pointer to the beginning of the Event Control Block chain is retrieved and examine until the expected event is found, if the current Event Control Block corresponds to the event, its state is examined, if it is activated then the state of the task in WAIT state is changed to a READY state, if the end of the Event Control Block chain is reached without finding the event, a system error is flagged.

[c35] o) Having a Priority Task Ordering subprocess which selects the task in the READY state that is to be switched to the ACTIVE state using the follow criteria the task priority and the amount of time in which a task has been in the READY state using the following steps:

i) retreiving the pointer from the beginning of the Task Control Block;

ii) examining all the tasks in the Task Control Block that are in the READY state;.

[c36] iii) intializing the max_priority, max_wait_time and task_id_winnner fields;
iv) checking sequentially all task states, if a task is not READY, the next task in the chain is checked, if a task is READY its priority is checked against Max_Priority, if the task's priority is higher, store task's data in the Max_Priority, Max_Wait_Time and Task_ID_Winner; if the task's priority is equal to the Max_Priority field, then compare the task's time on the WAIT state to the Max_Wait_Time field, if the current task's time on the WAIT state is higher, store task's data in the Max_Priority, Max_Wait_Time and Task_ID_Winner; and
v) repeating the prevoius step until the last link in the Task Control Block chain is examined. and
Having a Context and Control Restore subprocess which executes the restoration of all context variables associated with the active task and yields computer means control to a Parser, which continues the execution of the active task's instructions.